Service-Oriented Interactive 3D Visualization of Massive 3D City Models on Thin Clients

Dieter Hildebrandt, Jan Klimke, Benjamin Hagedorn, Jürgen Döllner
Hasso-Plattner-Institut
University of Potsdam, Germany
{dieter.hildebrandt|jan.klimke|benjamin.hagedorn|doellner}@hpi.uni-potsdam.de

ABSTRACT

Virtual 3D city models serve as integration platforms for complex geospatial and georeferenced information and as medium for effective communication of spatial information. In this paper, we present a system architecture for service-oriented, interactive 3D visualization of massive 3D city models on thin clients such as mobile phones and tablets. It is based on high performance, server-side 3D rendering of extended cube maps, which are interactively visualized by corresponding 3D thin clients. As key property, the complexity of the cube map data transmitted between server and client does not depend on the model's complexity. In addition, the system allows the integration of thematic raster and vector geodata into the visualization process. Users have extensive control over the contents and styling of the visual representations. The approach provides a solution for safely, robustly distributing and interactively presenting massive 3D city models. A case study related to city marketing based on our prototype implementation shows the potentials of both server-side 3D rendering and fully interactive 3D thin clients on mobile phones.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, distributed applications*; C.5.5 [**Computer System Implementation**]: Servers; D.2.11 [**Software Engineering**]: Software Architectures—*Service-oriented architecture (SOA)*; D.2.1 [**Software Engineering**]: Requirements/Specifications; I.3.2 [**Computer Graphics**]: Graphics Systems—*Distributed/network graphics*

General Terms

Algorithms, Design, Performance, Standardization

Keywords

Service-oriented architecture, mobile device, distributed geovisualization, 3D geovirtual environment, virtual 3D city model, 3D computer graphics

1. INTRODUCTION

3D geovirtual environments (3DGeoVEs) are a conceptual and technical framework for the integration, management, editing, analysis, and visualization of complex 3D geospatial information. *Virtual 3D city models* as a specialized and frequent type of 3DGeoVE serve as integration platforms for complex geospatial and georeferenced information. For application areas such as city planning and marketing, virtual 3D city models turned out to be effective means for the communication of planning related information, e.g., about land usage, transportation networks, public facilities, and real estate markets. Such systems have to provide up-to-date data, most efficient interaction capabilities, as well as effective, high-quality visual representations. Typically, the geodata required for representing virtual 3D city models in real world software applications have massive storage requirements. To give users interactive access to high-quality virtual 3D city models, the resources required by a 3D geovisualization system in terms of storage and computing capacity can be significant. This currently restricts the applicability of 3D geovisualization especially on mobile devices and for service-based and web-based systems.

Until today only "monolithic" geovisualization systems can cope with all these challenges of providing high-quality, interactive 3D visualization of *massive* 3D city models, but still have a number of limitations. Such systems typically consist of a workstation that is equipped with large storage and processing capabilities, as well as specialized rendering hardware and software, and is controlled by an expert who controls the virtual camera and decides which information to integrate into the visualization through a graphical user interface. Typically, only a single view is available on a single screen or projection; multi-user access and collaboration is usually not supported; and these systems mostly lack the emotional factor that is immanent to today's presentation and interaction devices such as smartphones and tablets. Often, such a system does not allow for easy and seamless integration of new or updated information, as data needs to be preprocessed to fit a specific internal format for enabling high-performance rendering. Furthermore, it may be difficult to adapt such an encapsulated visualization system to specific data and usages that require new, advanced visualization techniques. Even for today's high-performance visualization systems, it is a challenging task to combine the visualization of massive, large-scale 3D data with the visu-

alization in a high quality and high level of detail, e.g., with detailed facades or including indoor models. Due to their specific hardware configuration, such a system for 3D geovisualization cannot be moved physically ad-hoc and easily; but it requires time and effort to break it down, move it, and set it up again.

In addition to these drawbacks of monolithic 3D geovisualization systems, there is a strong demand to make virtual 3D city models interactively available on thin clients such as smartphones and tablets. While today, the usage of 2D maps on such devices is a common application, this has not been reached for massive, complex 3D geodata to the same extend. As a common solution, visualization systems can be deployed that distribute geodata and functionality over computers connected by a network using visualization clients (e.g., Google Earth). Thus, visualization clients and devices are required to provide locally only a fraction of the overall required resources while the major part is provided by the client's context. However, these techniques for 3D geovisualization have not been fully adapted to thin clients yet. 3D geovisualization for thin clients has to cope with barriers such as potentially limited bandwidth, limited rendering capabilities, and especially constrained energy resources.

In this paper, we present a system architecture for service-oriented, standards-based, interactive 3D visualization of massive 3D city models on thin clients such as smartphones and tablet computers that has the potential to overcome the aforementioned limitations of today's 3D geovisualization systems. It is based on a high performance, scalable, state-less 3D rendering service that implements the *Web View Service* (WVS) and provides sets of 2D images of projective views of 3DGeoVEs. Additionally, it is capable of organizing the geodata to be visualized based on layers, integrating thematic layers of raster or vector geodata into the visualization, and styling the visual representations according to declarative specifications. The visualization client interactively visualizes the 3DGeoVE based on *extended cube maps*, which it retrieves from the 3D rendering service. The client can control the visual representations through selecting desired layers and providing a styling specification when invoking the rendering service. As key property of the approach, the complexity of the cube map data transmitted between service and client and the rendering costs on client's side do not depend on the model's complexity. The use of service-orientation and standards facilitates designing distributed 3D geovisualization systems that are open, interoperable, and easily adaptable to changing requirements. The approach provides a solution for safely, robustly distributing and interactively presenting massive 3D city models. A case study related to city marketing based on our prototype implementation shows the potentials of both server-side 3D rendering and fully interactive 3D thin clients running on touch-enabled devices such as smartphones.

The contribution of this paper is to identify requirements and conceptually layout a system architecture for service-oriented, standards-based, interactive 3D visualization of massive, complex 3D city models on thin clients, to describe corresponding server and client architectures and implementations, and to evaluate the overall system architecture based on a case study. The remainder of this paper is structured as follows: Section 2 describes related work. Section 3 introduces requirements of the desired service-oriented 3D geovisualization system. Section 4 describes the overall system architecture for service-oriented, image-based visualization of and interaction with massive 3D city models. Section 5 provides details of a prototype service implementation. Section 6 presents details of a corresponding client implementation. In section 7, we provide preliminary evaluation results. Section 8 evaluates the architecture and implementation based on a case study. Finally, section 9 concludes the paper.

2. FUNDAMENTALS & RELATED WORK

The *service-oriented architecture* (SOA) paradigm promotes the idea of assembling application components into a network of *services* that can be loosely coupled to create flexible, dynamic business processes and agile applications that span organizations and computing platforms [22]. For categorizing services in a SOA, a common set of categories consists of *data*, *functionality*, *process*, and *interaction* each defining a layer in a distributed layered software architecture (Fig. 1). In the geospatial domain, the *Open Geospatial Consortium* (OGC) adopted the SOA paradigm and proposes standards for service interfaces, data models, and encodings. For the presentation of information to humans, the OGC proposes stateless *portrayal services*. For 3D portrayal, the *Web 3D Service* (W3DS) [24] and the *Web View Service* (WVS) [11, 10] are proposed as different approaches that are both still in the early stages of the standardization process. The W3DS delivers scene graphs that can be rendered by a client, whereas the WVS delivers rendered images of projected views that are ready for display. In addition, the WVS provides thematic *image layers* storing information such as color, depth, object ID, surface normal, and mask encoded in standard formats for a specified virtual camera specification (Fig. 3). This concept is based on the *G-buffer* [23] concept from 3D computer graphics. Complementary, the *Styled Layer Descriptor* (SLD) and *Symbology Encoding* (SE) [20] are standardization proposals for user-defined styling of 3D visual representations. Portrayal services may include support for SLD. The SLD and SE follow the most common way to implement styling in the mapping stage of the visualization pipeline (*M-styling*). In addition, we propose to implement styling after the rendering stage by *image post-processing* (*R-styling*) offering different trade-offs [12]. In both cases, clients specify styling in a declarative, rule-based styling language. For managing and providing geodata in different representations, the OGC proposes services such as the *Web Coverage Service* (WCS), *Web Map Service* (WMS), and *Web Feature Service* (WFS). *Web Processing Services* (WPS) can provide arbitrary functionality through a generic interface.

For distributed visualization systems, most commonly visual representations based on scene graphs, geometry such as triangle meshes, and textures are proposed and applied (such as in the W3DS [24], Google Earth, Microsoft Bing Maps 3D). Here, we focus on image-based representations since we estimate that they have the potential to support better the stated requirements (Sec. 3). For interactive, image-based, distributed visualization, the proposed approaches include streaming videos of rendered 3D models from a server to a tightly coupled client [18], applying *image-based model-*

ing and rendering (IBMR) [25] and warping a representation based on color and depth images retrieved from a remote server for rendering novel views [4], applying *point-based modeling and rendering* (PBMR) [9] and utilizing remotely rendered color and depth images as input for client-side PBMR [6], and rendering novel views on the client by warping between image-based panoramas retrieved from a server [5]. In addition, proposals exist for designing visualization systems as distributed systems (e.g., [3]), distributed systems based on SOA (e.g., [27]), or distributed systems based on SOA and OGC standards (e.g., [13, 2]). However, to the best of our knowledge, we are not aware of related work that proposes designing 3D geovisualization systems based on SOA, OGC standards, and image-based representations with the aim of meeting the stated requirements (Sec. 3). In particular, related work often does not address at the same time improving integration by loose coupling, interoperability, supporting lightweight clients, and the application to 3DGeoVEs.

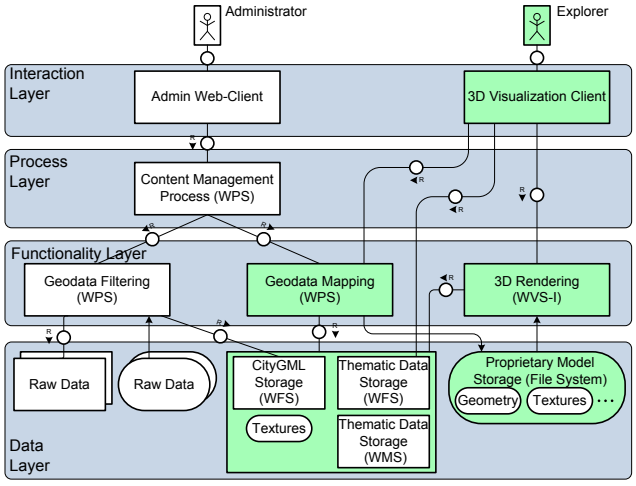

Figure 1: Overview of the system architecture.

3. REQUIREMENTS

In this section, we identify a set of requirements for 3D geovisualization systems. This particular set is valid for a specific, practically relevant class of 3D geovisualization systems and is informed by existing literature [14]. We place a particular focus on 3DGeoVEs and virtual 3D city models. Furthermore, we focus on 3D rendering for generating 2D images of projective views of primarily static CAD-based models with real-time interaction and navigation using six degrees of freedom.

Support for *integration* is required to connect computer systems effectively and efficiently on different levels of abstraction such as data, functionality, process, visualization, interaction, and system. It should improve the flexibility and efficiency of adapting systems to changing requirements and ease the reuse of software components. *Interoperability* increases the effectiveness and efficiency of the integration on the different abstraction levels and can be improved by applying standards. In the geospatial and the geovisualization domain, insufficient interoperability has been identified as a major barrier for progress in the respective domain. Typically, in real world applications, systems are required to facilitate processing, visualizing and interacting with *massive amounts of geodata*. In particular, this applies to virtual 3D city models. Providing *effective, high quality visual representations* improves the effectiveness of a geovisualization system and is facilitated by advanced, complex, innovative visualization algorithms and in certain cases massive amounts of data (e.g., for virtual 3d city models), for both realistic and abstract views. Support for *platform independency* comprises the relative independency of a system solution from software and hardware platforms on different levels of abstraction and adaptive and moderate use of platform resources. It can improve dissemination and reduce costs. A *high degree of interactivity* is a key defining characteristic as well as a crucial requirement for geovisualization systems and should be effective and efficient. Support for *styling* visual representations allows for controlling what (e.g., filtering of features) and how to portray (e.g., mapping of features to geometries and visual attributes) and is essential for interaction and generating different visualizations from the same base data. A distributed geovisualization system has to *scale* according to the number of users working with the system and the size and complexity of the underlying geodata.

4. SYSTEM ARCHITECTURE

In this section, we present an architecture for a service-oriented, standards-based, distributed system that implements a specific *configurable, service-oriented geovisualization pipeline* for 3DGeoVE (Fig. 1). The main purpose of the system is to enable a user with a lightweight client (e.g., web browser, mobile application) to explore interactively a massive, static, virtual 3D city model through the Internet based on server-side 3D rendering. Additionally, a user can control the contents and styling of the visual representations by selecting from predefined lists of layers to include in the visualization (including thematic layers based on vector data), a style for each selected layer (including styles based on thematic raster data), and a post-processing style that can affect the whole scene across layers. To facilitate these purposes, the system offers functionality for importing and preprocessing the input geodata that comprises the virtual 3D city model into a representation that is ready to be rendered. In this section, we present an overview of the proposed architecture. In the following two sections, we focus on and present details on two components of this architecture: the 3D rendering service (Sec. 5) and the 3D visualization client (Sec. 6).

In the filtering stage, the purpose of the WPS-based `Geodata Filtering` service is to transform raw geodata into an integrated, semantic representation of a virtual 3D city model. For this, the service first imports raw geodata (e.g., shape files from the cadastre, digital terrain models, aerial photographs, CAD building models) from potentially different external organizations. Then, it transforms the raw data into a model adhering to the CityGML [26] data model and texture images referenced by the CityGML data model. The CityGML representation is then transferred to and stored by a `CityGML Storage` service that is based on a Web Feature Service (WFS). The texture images are stored separately on a file server. The `Geodata Filtering` service is implemented in C++ based on *OpenSceneGraph* (OSG, `www.`

openscenegraph.org). OSG is used for this and other services (see below) since it already provides considerable functionality for processing, rendering, importing and exporting geometry, texture and scene graphs in different data formats. For the importer, the rendering functionality of OSG is not used and extensions are implemented for accepting WPS requests, reading shape files, and exporting CityGML. The services for Raw Data and textures are implemented as file servers. The CityGML Storage service is implemented using the Java-based WFS provided by deegree (www.deegree.org) and a 3D geo database for CityGML [26] based on Oracle 10G R2.

In the mapping stage, the WPS-based Geodata Mapping service transforms the standards-based CityGML and texture representations into proprietary, internal representations optimized for 3D rendering. For this, the CityGML and texture representations are retrieved from the respective services, processed, and stored by a Proprietary Model Storage service in proprietary data models. The 3D Rendering service provides 2D images of projected views of 3DGeoVEs (e.g., virtual 3D city models). For this, the service retrieves data from the Proprietary Model Storage, accepts requests from service consumers containing virtual camera specifications and SLD styling specifications, and generates and returns 2D images. Additionally, the 3D Rendering service accesses the CityGML Storage service to satisfy clients requesting additional properties of portrayed features. The Geodata Mapping and 3D Rendering services are implemented in C++ based on OSG and associated projects (such as OSG Earth, VirtualPlanetBuilder, GDAL) with specific extensions including components for importing and exporting the respective data formats and implementing the respective WPS and WVS interfaces.

The architecture defines two user roles and their tasks: Administrator and Explorer. An Administrator uses the Admin Web-Client for content management (CM) tasks. The outlined architecture only supports the basic task of importing raw geodata into the system and preprocessing it into a representation ready to be rendered. Based on inputs from the administrator, the Admin Web-Client invokes the WPS-based Content Management Process service. This service uses the Geodata Filtering and Geodata Mapping services for the implementation of its process and is capable of handling occurrences of long download durations, network failures, invalid or corrupt data and so on. The Admin Web-Client is implemented as a browser-based web application. The Content Management Process is implemented using the Java-based WPS provided by deegree. An Explorer uses the 3D Visualization Client for interactively exploring the virtual 3D city model. In a short, closed loop, the 3D Visualization Client requests images from the 3D Rendering service, displays the images, and requests new images based on the users input. The 3D Visualization Client is implemented using C++, Objective-C, OSG and OpenGL ES for running on the iOS platform (smartphones, tablet computers), and using Java, JOGL and OpenGL for executing as a Java applet inside web browsers.

5. 3D RENDERING SERVICE

In this section, we describe the 3D Rendering service. Its primary purpose is to provide 2D images of projected views of 3DGeoVEs. The primary operation of the service is the GetView operation. A *view* can be defined by a virtual camera specification, a list of requested image layer and their requested encodings, a list of requested geodata layer to include in the visualization, and styling specifications. To reduce network round trip times, a service consumer can request several views each consisting of several image layers with one call of the operation. In the following, we describe the software components and general workflow of the service (Fig. 2).

5.1 Service Endpoint

The Web Server initially receives service requests from service consumers. To allow the service to process a predefined number r of requests concurrently, the Web Sever creates at most r threads for executing instances of the WVS Endpoint, WVS GetView, and Encoder. A request is either passed directly to an available WVS Endpoint or queued for later execution. The WVS Endpoint parses the request according to the WVS specification [10] and calls the appropriate operation handler. The WVS specification proposes several of operations. Here, we focus on the GetView operation. Moreover, the specification defines the WVS as a stateless service. The WVS GetView performs parsing on the request specific to the operation and passes it to the Render Master.

5.2 Parallel Processing of Requests

The purpose of the Render Master is to distribute the workload represented by incoming requests to resources available for processing the requests represented by several Render Workers. The Render Master and Web Server are the two primary components of the architecture that perform process synchronization for the purpose of managing shared resources. The major goals in both cases are optimizing resource utilization, maximizing throughput, and minimizing response times. For achieving these goals, the Render Master proceeds as follows. First, it is capable of receiving requests concurrently from different WVS GetView instances. Incoming requests are queued. Either if at least one Render Worker is already available for processing or as soon as at least one becomes available, the next available request is taken from the queue. When processing a request, it is split into a set of *tasks*. Each Render Worker can process exactly one task at a time. For load balancing, we propose the following algorithm. First, for each view in a request a task is created. If the number of tasks of the current request t is equal to the number of available worker w, then each task is assigned to an available worker. If $t > w$, then the tasks are processed iteratively: Tasks are assigned to available workers until no more worker is available. When a worker has finished a task from the current or a previous request, it is assigned the next task from the current request. Workers that become available are always assigned to tasks from the current request. The next request is not pulled from the request queue until each task of the current request was or is assigned to a worker. The iteration stops when all tasks of the request have been processed. If $t < w$, then two options exist. The first option is to assign each task to a worker. This allows to pull further requests from the request queue and to start processing them with the available worker. The second option is to split some tasks into smaller tasks not exceeding w in number. The split and the unsplit tasks are then assigned to the workers. This option allows to compute

a single request faster. The heuristic for choosing an option takes into account factors such as the current length of the request queue and the estimated response time for the current request in the split and unsplit case. For splitting tasks into smaller tasks, we use *sort-first decomposition* [19]. Each view (represented by the original task) is split in the view plane into a set of rectangular 2D tiles (represented by the split tasks replacing the original task) covering the complete view.

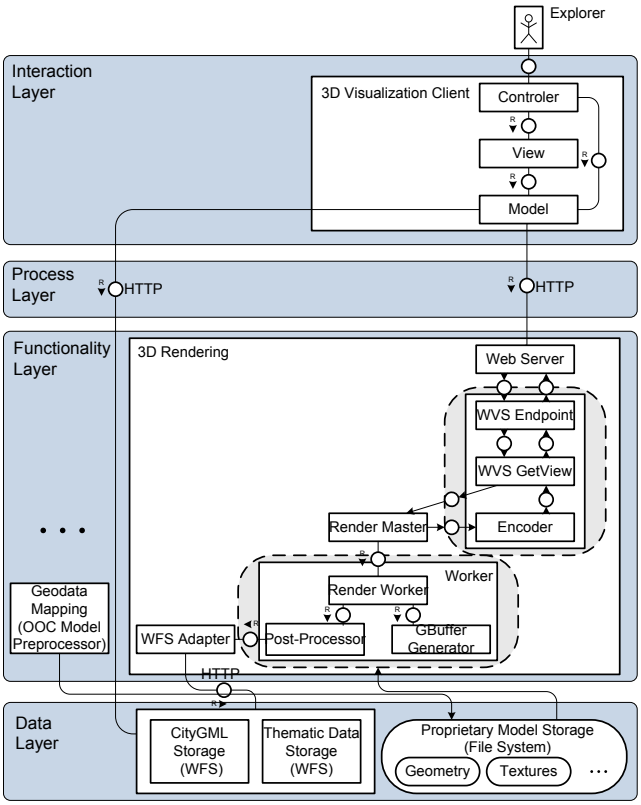

Figure 2: Architecture of the 3D rendering service and the 3D visualization client.

For deciding which task to assign to which available worker and for deciding when and how to split tasks, the `Render Master` can estimate how much time a given worker would take for processing a given task. We assume that the amount of time a specific worker takes for processing a task is a function of its hardware and software configuration, the current task, and the history of already processed tasks. The latter is of importance since we assume that the 3DGeoVE is massive and exceeds the main memory and GPU memory capacity of a worker. Thus, we use *out-of-core* (OOC) rendering techniques [8] that store the complete data of a 3DGeoVE on external storage and keep only a fraction of it required for rendering at least the current task in memory. Therefore, typically, a worker can process a rendering task faster, if data for previous tasks is still in memory and caches and can be reused for the current task. Following these assumptions, a function estimates the time a given worker would take to process a given task using a heuristic. This function is then used, e.g., to compute for a number of tasks and available workers a configuration of assignments that minimizes the maximum response time.

A `Render Worker` processes tasks assigned to it from the `Render Master`. Each `Render Worker` has its own thread of execution and a GPU associated to it. Either it executes on the same CPU as the `Render Master`, on a different CPU on the same computer or a different computer connected via a network. Each task processed by the `Render Worker` represents a view of the 3DGeoVE. A set of tasks can be the result of a sort-first decomposition of an original task. Each view consists of one or more image layer. For decomposing tasks, we use the sort-first scheme instead of other schemes such as *sort-last* [19] in order to be able to execute the post-processing of a view on the same worker that generated the view. This has several advantages. First, not only the rendering and G-buffer generation but also the post-processing, which can also be costly to compute [12], can execute concurrently on workers. In addition, generated G-buffers still reside in GPU memory and can be used in place for post-processing. Finally, temporary G-buffers created only for specific post-processing effects never have to be transferred from the GPU memory and can be discarded after use. In contrast, when using, e.g., sort-last decomposition, post-processing of the view cannot be executed before all its parts are transferred (over the network or via main memory) and the final view is composited. When building or splitting tasks, the image layers requested for a view are never distributed to different tasks. Tasks are never split along image layers. This has several advantages. First, multiple image layers for a single view can be most efficiently created on the same worker in one rendering pass using the GPU technique *multiple rendering targets*. Additionally, since OOC rendering techniques must be applied, the data required for generating an image layer must be transferred from the external storage prior to rendering. This data can be reused for generating multiple image layers. Finally, since post-processing might require the requested image layers, they can be made available efficiently for this purpose directly on the GPU.

5.3 Rendering and Post-Processing

When receiving a task, the `Render Worker` first invokes the `Post-Processor` for preparing the task for the subsequent post-processing, then invokes the `GBuffer Generator` for generating the image layers by rendering the 3DGeoVE as specified in the task, and, finally, invokes the `Post-Processor` again to execute effectively the post-processing on the image layers. The purpose of the `Post-Processor` is to style images of the 3DGeoVE according to a given product-specific, company-specific cartographic and thematic design. The `Post-Processor` implements styling by post-processing images [12] (R-styling). The `Post-Processor` accepts as input a styling specification and a set of image layers representing a view of the 3DGeoVE. The output consists of a set of image layers representing a styled view (Fig. 3). The styling language offers a range of operators for, e.g., data integration, feature abstraction, increasing photorealism (e.g., by global illumination), and focus and context visualization. In particular, operators are provided for projecting thematic raster data on the geometry of the 3DGeoVE, and blending different *level of abstraction* (LOA) [7] represented as geodata layer seamlessly in image space. When executing styling specifications, the `WFS Adapter` is used to query properties of the original geodata (e.g., thematic attributes) no longer available in the `Proprietary Model Storage`. For processing

a given styling specification, the Post-Processor typically requires a set of image layers with specific types. If image layers are required internally for styling but are not required from the external service consumer as output, then the Post-Processor adds these image layers to the current task before rendering and disposes them again after the post-processing.

The purpose of the GBuffer Generator is to generate a sequence of image layers by rendering the 3DGeoVE as specified in a task. This component accepts as input a task that specifies a virtual camera specification, a list of requested image layer, a list of requested geodata layer, and styling specifications. The output is a sequence of generated image layers. To render massive 3DGeoVE efficiently, the geodata representing the 3DGeoVE has to be transformed into a representation optimized for 3D rendering. Since this representation can exceed main memory capacities, the optimized representation is stored on external storage and only parts of it required for at least processing the current task are kept in main memory.

The Geodata Mapping service (Sec. 4) is responsible for transforming geodata into a representation that is styled and optimized for efficient 3D rendering. The Geodata Mapping service accepts as input references to geodata in standardized representations, a specification for controlling how the service divides the geodata into geodata layers containing sets of features, a styling specification for controlling how the service maps geodata to geometries and appearance attributes (M-styling). As output, the service writes to the Proprietary Model Storage a representation optimized for rendering by the 3D Rendering service. The styling language offers basic operators (e.g., for defining polygon material properties) and high-level operators such as computing LOA representations for building models [7]. Additionally, the Geodata Mapping service supports managing the data contained in the Proprietary Model Storage. It supports adding new data to already existing data, updating or deleting existing data, and storing and managing multiple versions of data that resulted from transforming the same original geodata with different styling specifications. The access to the data in the Proprietary Model Storage is synchronized allowing the 3D Rendering service to serve requests while the Geodata Mapping service is updating the underlying data concurrently.

The data optimized for 3D rendering is organized in OOC, hierarchical, multi-resolution data structures [1]. For efficient rendering, we distinguish the following types of data and organize them in specific, separated, interlinked data structures: terrain geometry (quadtree), terrain textures (clipmaps), features with unique geometry such as buildings, transportation, and water bodies (quadtree for geometry, virtual textures), and features with instanced geometry such as city furniture and vegetation (quadtree for storing references to instanced geometry, texture atlases). Moreover, we follow general principles that are crucial for efficient rendering such as drawing geometry in large batches and minimizing GPU state changes (aggregating features in vertex buffer objects, texture atlases, state sorting), and using level of detail representations (LOD, hierarchical simplified geometry in the quadtree, static LOD versions for instanced geometry, texture mipmaps). For rendering, the GBuffer Generator traverses the hierarchical data structures,

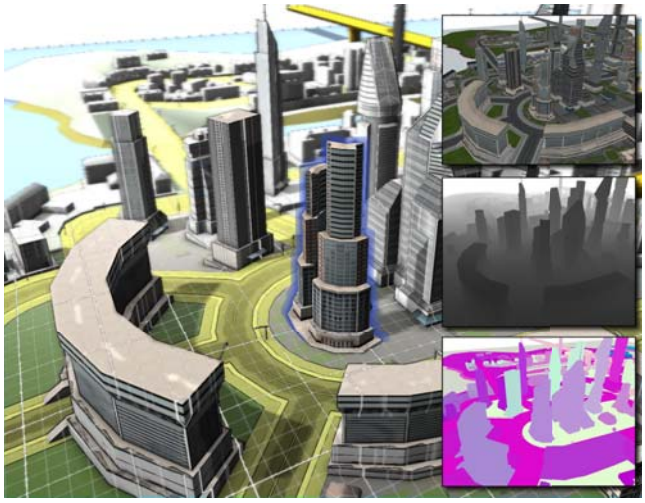

Figure 3: Example image layer provided by the 3D rendering service: color image layer with advanced, post-processing-based styling (center), plain color, depth, object ID image layer without styling (inset top to bottom).

identifies the appropriate LOD of geometries and textures that contribute best to the current view and that are part of requested geodata layer, retrieves data from disk that is required but not yet loaded, and, finally, draws geometries using referenced textures writing in parallel to multiple render targets. Each render target represents a requested image layer. The generated image layers are returned as output. In contrast to common real-time rendering, here, we expect far less frame-to-frame coherence from one requested view to the next making common optimization techniques less effective. In addition, when data is required for rendering but is not yet loaded form the external storage, the rendering cannot finish until all required data has been loaded.

5.4 Composition and Encoding of Results

For each request, the Render Master collects the results of the tasks the request was split into from the individual Render Workers. If the Render Master split a task and its view by sort-first decomposition, then now it composes the complete view from the results of the split tasks. To exploit parallelism, once the Render Master has completed the work on an individual image layer, it is immediately passed to the Encoder of the thread the request originated from for further processing. Thus, available image layers as results of a request are already passed to the next stage of the processing pipeline while other image layers might still be worked on by Render Workers.

The Encoder receives as input an individual image layer and an encoding specification. Conceptually, each sample value (e.g., depth) in an image layer is first transformed from a computer graphic (e.g., nonlinear, normalized) to a geographic coordinate system (e.g., linear, meters), and then into a compressor specific representation (e.g., 32bit float value masked as RGBA). Subsequently, the image layer is compressed (e.g., LZ77 and Huffman coding) and encoded in a standardized format (e.g., PNG). Since the straight-

forward compression of depth images using standard image compression encodings [10] proves to be far less efficient than it is with different image layer types [11], we offer as an alternative the encoding of depth images as adaptively triangulated meshes (*depth mesh*, or *DMesh* [21]) encoded using the X3D standard. Each encoded image is passed in turn to the `WVS GetView` and the `WVS Endpoint` components. The `WVS Endpoint` streams each image layer over an HTTP outbound socket stream to the service consumer once an image is received. Thus, parts of the result are already streamed to the service consumer even if the complete result is not yet computed.

6. 3D VISUALIZATION CLIENT

In this section, we present the `3D Visualization Client` as introduced in section 4 (Fig. 2). The purpose of the client is to enable a user to explore and interact with a remote, massive 3DGeoVE rendered by a `3D Rendering` service implementing the WVS interface through the Internet. The client retrieves sets of 2D images of projective views of the 3DGeoVE from the service. The fundamental challenge when using image-based representations are to provide a high degree of interactivity while making efficient use of the network channel in the course of interactions, and to put low hardware requirements on the client. We propose three different concepts for implementing clients that address these challenges with different trade-offs [15]. The concepts differ in how they exploit images retrieved from the service and use additional service-side functionality. The concepts are based on 1) image retrieval and display, 2) image-based modeling and rendering (IBMR), and 3) point-based modeling and rendering (PBMR). In this paper, we focus on a client based on IBMR (Fig. 4).

The client requests cube maps for user specified virtual camera locations in several information dimensions (color, depth, object ID) from the WVS. We refer to cube maps with multiple information dimensions as *extended cube maps* to differentiate them from the common concept of cube maps known from 3D computer graphics [1] that typically contain only a single information dimension, i.e., color. The cube maps are transferred as image sequences to the client. The client reinterprets the images as a description of the remote 3DGeoVE consisting of 3D surface patches with attributes including 3D position, color, and object ID. From the surface patches, 3D meshes are constructed. In case the WVS supports the DMesh encoding for depth images, the client requests this encoding and directly uses the DMesh as 3D mesh. The meshes represent the local, partial, approximated 3D reconstruction of the remote 3DGeoVE.

To support navigation and interaction, novel views can be rendered of the local 3D reconstruction from arbitrary virtual camera viewpoints in real-time with interactive frame rates. Thus, the client applies a latency hiding technique by rendering novel views from the 3D reconstruction with low latency while retrieving additional images from the WVS concurrently with high latency. The client supports interaction tools including common direct navigation techniques (e.g., rotate virtual camera, pan, zoom, and orbit), field of view-based zoom with selective refinement of the cube map, goto navigation with smooth virtual camera animation, selection and highlighting of object features (e.g., buildings) within the 3DGeoVE, retrieving and displaying additional thematic information for object features from the WVS, and measuring the Euclidean distance between arbitrary spatial positions.

7. EVALUATION

In this section, we discuss how the proposed `3D Rendering` service, the `3D Visualization Client`, and the general approach support meeting the requirements identified in section 3. Additionally, we present preliminary quantitative results of our initial implementations.

7.1 Discussion

The proposed `3D Rendering` service presented in section 5 supports meeting the requirements identified in section 3 as follows. Integration is supported by the concurrent integration of the new or updated geodata in the `Proprietary Model Storage`. Interoperability is supported by applying standards for the service interface and input and output data where applicable. It is currently limited by the fact that standards do not exist for efficiently representing massive geodata optimized for 3D rendering and image layer types other than color. Massive geodata is supported by concurrent data integration and applying OOC, LOD and general principles for efficient rendering. Effective, high quality visual representations are supported by the styling capabilities (M-styling and R-styling) and the offered powerful high-level operators such as the LOA operator and the capability to render massive geodata sets. A high degree of interactivity is supported by answering service requests with low latency by applying parallelism, efficient rendering algorithms and data structures, and by efficiently encoding image layers. Styling is supported by specific styling capabilities (M-styling and R-styling). Scalability is supported by exploiting parallelism when processing a request on different levels and by taking advantage of available hardware resources (e.g., multiple CPUs, GPUs).

As we have reported in [15], advantages of the `3D Visualization Client` based on IBMR (section 6) include that its implementation, hardware resource requirements, and integration efforts are only moderately complex, it effectively provides low latency interaction and display updates, supports several interaction techniques efficiently by exploiting the retrieved G-buffers from the WVS, reuses the images retrieved from the WVS over several frames rendered locally on the client, and displays results of WVS requests with changed styling specifications as soon as the limited set of locally maintained depth meshes and images with previously requested styling are evicted from local memory. On the other hand, aggregating locally a 3D reconstruction of visual representations of the remote 3DGeoVE based on depth meshes is not optimally effective and efficient, retrieved images can only be reused for a limited number of frames rendered locally, and the locally rendered novel views are approximations that suffer from hard to control under- and oversampling issues.

The general approach of building 3D geovisualization systems based on SOA and standards has specific potential and challenges. In [14], we present a discussion on this topic.

7.2 Quantitative Results

In the following, we present preliminary quantitative results of our initial, not yet optimized implementations of the `3D Rendering` service and the `3D Visualization Client` based on IBMR.

In the first experiment, we aim at measuring the rate at which the `3D Rendering` service can provide service consumers with rendered images. For this experiment, we created a service consumer that stresses the service by sending 40 requests to the service each requesting three image layers (color, depth, and object ID) for one 3D view. The service consumer sends up to 10 requests in parallel. The service processes each request sequentially and does not apply the techniques presented in section 5 for parallel processing and image post-processing. For each request, the service consumer receives one HTTP multipart response containing the three requested image layers. In total, the service generates and delivers 120 images. We measure the time for sending a request, rendering the images, compressing the images (JPEG, PNG), sending the images, and decompressing the images by the service consumer. The experiment is performed in an intranet environment. The service is executed on a desktop PC (Windows Server 2003, 1.86GHz double core, 2 GB RAM, nVidia GeForce GTX 260). The service consumer is executed on a different PC connected to the network. As a result, we measure that a service consumer can receive images at an average rate of 5.7 images per second for an image resolution of 512x512 and a 2.6 for 1024x1024. Second, we measure the memory size of generated and transferred image layers while navigating through the 3DGeoVE. For an image resolution of 512x512, on average, color required 77.95 kbytes (JPEG), depth 199.63 kbytes (PNG), and object ID 9.56 kbytes (PNG). We expect to achieve higher delivery and compression rates in the future by applying the proposed parallel processing and compression techniques.

Third, we measure the rendering rate of the Java-based implementation of the `3D Visualization Client` based on IBMR. The client is executed in a web browser on a notebook (Windows XP, 2.4GHz double core, 3 GB RAM, nVidia Quadro FX 570M with 512 MB RAM). In this experiment, we log the rendering rate of the client while a user navigates several minutes through the 3DGeoVE using different navigation techniques. While the user navigates, the client retrieves images (512x512) from the service as appropriate. Depth meshes are not adaptively triangulated. The average rate of frames per second is zero when the user is not interacting with the client and the current view does not change since the 3D view is not updated in this situation, 284 when the user looks around from a fixed camera position (rendering a cube map), 102 when the user employs a fly navigation technique (rendering few depth meshes), and 71 when the user uses a goto navigation technique (rendering up to 12 depth meshes). We expect to achieve higher rendering rates in the future by applying adaptively triangulated depth meshes.

8. CASE STUDY: MOBILE BERLIN 3D

The 3D city model of Berlin is currently one of the largest, textured city models available (Fig. 4). It covers the complete urban area of the city of Berlin including over 500.000 buildings in CityGML LOD2, 350 buildings in LOD3 or LOD4, and more than 3 million single textures. A multitude of geodata sets exists that provide additional information (e.g., public transport routes, land value data, solar potential) through WMS or WFS services. Interactively visualizing this massive city model puts significant requirements on a systems hardware and software.

Figure 4: The official 3D city model of Berlin (center). Screenshot of the client application executing on the Apple iPhone (inset).

For *Berlin city marketing* (`www.businesslocationcenter.de`), there are two major applications for 3D visualization: 1) face-to-face meetings where companies evaluate city locations using additional data integrated in the visualization guided by city marketing staff, 2) presentations on exhibitions or trade fairs to advertise Berlin as a location for business. In both cases, the integration of additional geodata into the visualization of the 3D city model is significant for decision-making and presentation. The proposed distributed system overcomes various limitations of the currently employed monolithic system. By adding individual interactive views on a client device for each participant of a meeting to a single shared view for all participants, the efficiency of communicating geoinformation can be improved by computer-assisted collaboration. In addition, individually controlled views on client devices coupled with touch-based interaction have the potential to make the experience more personal and emotional. For the second application case, the availability of the visualization system in differing remote locations is required. With the proposed distributed system, the cost and time intensive effort to move physically the systems hardware is replaced by moving lightweight client devices and supplying a robust Internet connection to the services performing the major part of the resources required for visualization.

The proposed system enables additional valuable applications such as creating visualization clients customized for specific use cases such as giving talks at remote locations, using the system as a foundation for creating collaboration tools for specific use cases [17], and supplying business contacts with tailored visual representations of the city model augmented with thematic geodata, e.g., via URLs provided in emails.

9. CONCLUSIONS

In this paper, we described challenges for the visualization of and interaction with massive 3D city models and the limitations of today's 3D geovisualization systems. Driven by the demand for interactive, effective, high-quality 3D geovisualization on thin clients (e.g., smartphones and tablets) we presented a system architecture that copes with these challenges in a service-oriented, standards-based way. The approach is based on high performance, server-side 3D rendering of extended cube maps, which are interactively visualized by corresponding 3D thin clients. As key property, the complexity of the cube map data transmitted between server and client does not depend on the model's complexity. The proposed architecture provides a flexible mechanism for styling of 3DGeoVEs, which allows to adjust effectively their visual representation to the requirements of specific use cases. In addition, the approach ensures the security of the underlying geodata and investments of its owner by transferring only images to clients instead of geodata or computer graphic data. The proposed system architecture has been partly implemented and demonstrated by the implementation of a WVS and two implementations of the 3D visualization client, one built for smartphones and tablet computers and a second for web browsers. The client application uses touch-based interaction techniques for controlling the virtual camera and exploits the emotional factor of devices with tangible displays. For a city marketing use case, we discussed the benefits of the proposed, flexible system in comparison to current monolithic systems.

The proposed system architecture represents a solution to cope effectively with the increasing demand for flexible access to and high-quality visualization of massive 3D geodata that is driven by the ongoing and increasing digitalization of the real world. Through the systematic separation of concerns introduced by the service-oriented approach, new developments in client hardware and software (e.g., in-memory approaches) as well as new trends in specialized rendering servers, fast storage systems and fast networks can be exploited. The described approach promises to make visual representations of massive 3D geodata available in an interactive manner and in a high quality anytime and anywhere. It could not only support existing but could also lead to new applications, systems and business models, e.g., in the areas of edutainment, personal navigation, or collaborative systems for public participation in urban planning. Moreover, the combination of the described service-oriented approach with the resource elasticity promised by the cloud computing paradigm could form the basis for new business models that require efficient, high-quality and low-cost deployment of massive 3D geodata.

Future work includes establishing and implementing styling as a functional component of the outlined service-oriented system architecture, to consider dynamic data throughout the distributed visualization process (e.g., sensor data and animations), to further cope with and overcome the interaction barriers that come along with the image-based approach, and to extend the implementation for collaborative systems including, e.g., sketch-based annotations within the presented 3DGeoVEs [17, 16].

10. ACKNOWLEDGMENTS

This work has been partly funded by the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group 3D Geoinformation (`www.3dgi.de`). We also thank Berlin Partner GmbH and Berlin's senat department for urban development for supporting our work, the 3D Content Logistics GmbH (`www.3dcontentlogistics.com`) for inspiring discussions on the topic, and Autodesk Inc. for successful collaboration on the visualization client.

11. REFERENCES

[1] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., Natick, MA, USA, third edition, 2008.

[2] J. Basanow, P. Neis, S. Neubauer, A. Schilling, and A. Zipf. *Towards 3D Spatial Data Infrastructures based on Open Standards*. Lecture Notes in Geoinformation and Cartography. Springer, 2008.

[3] K. Brodlie, D. A. Duce, J. R. Gallop, J. P. R. B. Walton, and J. Wood. Distributed and Collaborative Visualization. *Computer Graphics Forum*, 23(2):223–251, 2004.

[4] C.-F. Chang and S.-H. Ger. Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In *Proceedings of the Third IEEE Pacific Rim Conference on Multimedia (PCM 2002)*. Springer, 2002.

[5] D. Filip. Introducing smart navigation in Street View. `http://google-latlong.blogspot.com/2009/06/introducing-smart-navigation-in-street.html`, 2009.

[6] J. Ge. *A Point-Based Remote Visualization Pipeline For Large-Scale Virtual Reality*. PhD thesis, University of Illinois at Chicago, 2007.

[7] T. Glander and J. Döllner. Abstract Representations for Interactive Visualization of Virtual 3D City Models. *Computers, Environment and Urban Systems*, 33(5), 2009.

[8] E. Gobbetti, D. Kasik, and S.-e. Yoon. Technical Strategies for Massive Model Visualization. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 405–415, New York, NY, USA, 2008. ACM.

[9] M. Gross and H. Pfister. *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., 2007.

[10] B. Hagedorn, editor. *Web View Service Discussion Paper, v0.6.0*. Open Geospatial Consortium Inc., 2010.

[11] B. Hagedorn, D. Hildebrandt, and J. Döllner. Towards Advanced and Interactive Web Perspective View Services. In *Developments in 3D Geo-Information Sciences*, Lecture Notes in Geoinformation and Cartography. Springer, 2009.

[12] D. Hildebrandt. Towards Service-Oriented, Standards- and Image-Based Styling of 3D Geovirtual Environments. In C. Meinel, H. Plattner, J. Döllner, M. Weske, A. Polze, R. Hirschfeld, F. Neumann, and H. Giese, editors, *Proceedings of the 5th Retreat of the HPI Research School*. Universitätsverlag Potsdam, 2011.

[13] D. Hildebrandt and J. Döllner. Implementing 3D Geovisualization in Spatial Data Infrastructures: The Pros and Cons of 3D Portrayal Services. In W. Reinhardt, A. Krüger, and M. Ehlers, editors, *Geoinformatik 2009*, volume 35. ifgiprints, 2009.

[14] D. Hildebrandt and J. Döllner. Service-oriented, standards-based 3D geovisualization: Potential and challenges. *Journal on Computers, Environment and Urban Systems*, 34(6):484–495, 2010. GeoVisualization and the Digital City - Special issue of the International Cartographic Association Commission on GeoVisualization.

[15] D. Hildebrandt, B. Hagedorn, and J. Döllner. Image-Based, Interactive Visualization of Complex 3D Geovirtual Environments on Lightweight Devices. In *7th International Symposium on LBS and Telecartography*, 2010.

[16] J. Klimke and J. Döllner. Geospatial Annotations for 3D Environments and their WFS-based Implementation. In *Geospatial Thinking*, Lecture Notes in Geoinformation and

Cartography, pages 379–397, Berlin, Heidelberg, 2010. Springer.

[17] J. Klimke and D. Jürgen. Combining Synchronous and Asynchronous Collaboration within 3D City Models. In *Proceedings of GIScience 2010*. Springer, 2010.

[18] F. Lamberti and A. Sanna. A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):247–260, 2007.

[19] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics and Applications*, 14:23–32, 1994.

[20] S. Neubauer and A. Zipf, editors. *3D-Symbology Encoding Discussion Draft, Version 0.0.1*. Open Geospatial Consortium Inc., 2009.

[21] R. Pajarola, M. Sainz, and Y. Meng. DMesh: Fast Depth-Image Meshing And Warping. *International Journal of Image and Graphics*, 4(4):653–681, 2004.

[22] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *Computer*, 40(11):38–45, 2007.

[23] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. *SIGGRAPH Computer Graphics*, 24(4):197–206, 1990.

[24] A. Schilling and T. H. Kolbe, editors. *Draft for Candidate OpenGIS Web 3D Service Interface Standard, v0.4.0*. Open Geospatial Consortium, 2010.

[25] H.-Y. Shum, S.-C. Chan, and S. B. Kang. *Image-Based Rendering*. Springer, 2007.

[26] A. Stadler, C. Nagel, G. König, and T. H. Kolbe. Making interoperability persistent: A 3D geo database based on CityGML. In J. Lee and S. Zlatanova, editors, *Proceedings of the 3rd Intl. Workshop on 3D Geo-Information*, Lecture Notes in Geoinformation & Cartography. Springer, 2008.

[27] H. Wang, K. W. Brodlie, J. W. Handley, and J. D. Wood. Service-oriented approach to collaborative visualization. *Concurrency and Computation: Practice & Experience*, 20(11):1289–1301, 2008.